Introduction to Artificial Intelligence

丁尧相
浙江大学

Fall & Winter 2022
Week 10

Announcements

- We will release Problem set 3.1 after this lecture.
- Problem set 2 is due on next Monday.
- We will release Lab project 2 this week.
- We will release a template for Lab project 1 this week.

Machine Learning: I

- Basic concepts
- Nearest neighbor classification
- Linear classification
- Bias-Variance Trade-Off & Regularization
- Take-Home Messages

Machine Learning: I

- Basic concepts
- Nearest neighbor classification
- Linear classification
- Bias-Variance Trade-Off & Regularization
- Take-Home Messages

Machine Learning

In machine learning, we want to obtain complex programs (functions) by learning from experience instead of programming by hand.

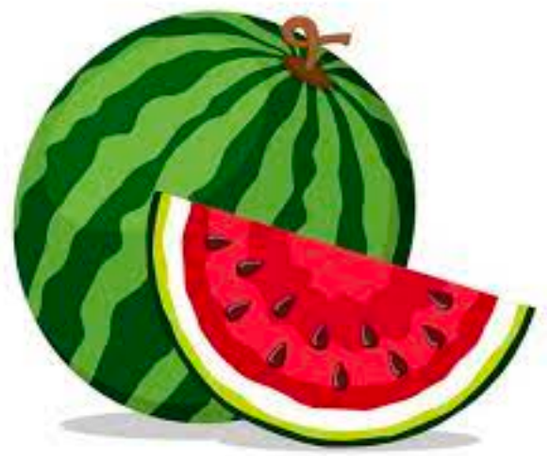
Computer Program

Input	→	Output
numbers		sorts
graph		shortest path
random seed		pseudo-random numbers
items: {weight, value}		knapsack solution
images		classes
game		actions to play

Example: Watermelon Classification

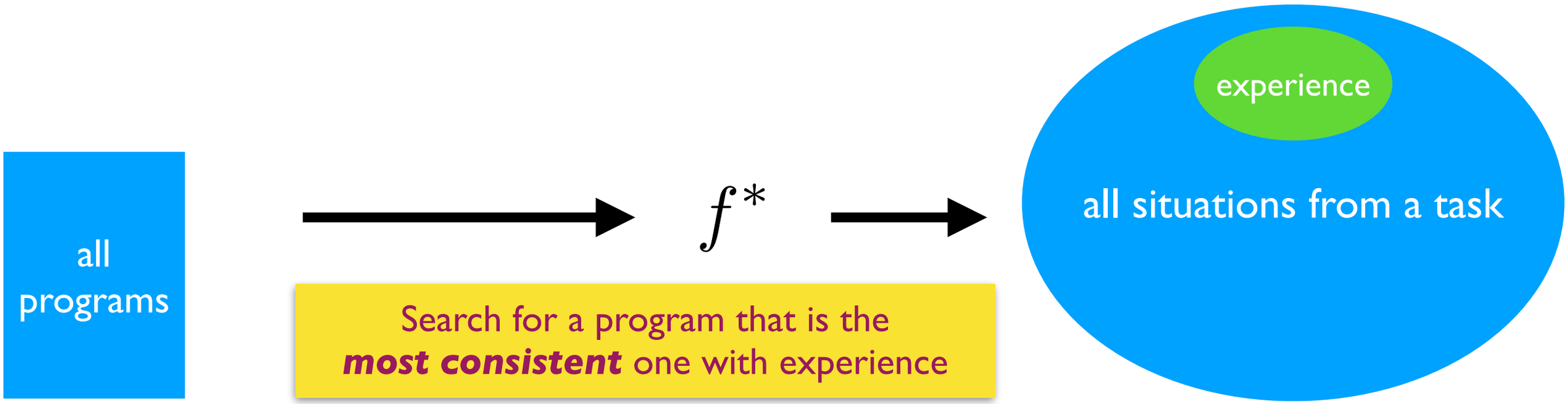
- Build a program to detect whether a watermelon is good or not.
- The program: $f : X \to Y$
  - X : features of watermelon, e.g. color, root type, touch sound.
  - Y : binary variable, indicating good or bad.
- Dataset: instances of (X, Y) pairs:

编号	色泽	根蒂	敲声	好瓜
1	青绿	蜷缩	浊响	是
2	乌黑	蜷缩	浊响	是
3	青绿	硬挺	清脆	否
4	乌黑	稍蜷	沉闷	否



The target is to find the best f^* which has the highest classification accuracy.

Machine Learning



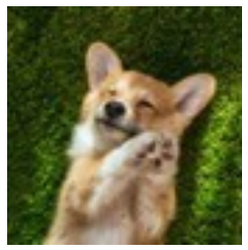
"Machine learning = task + data + objective + algorithm".

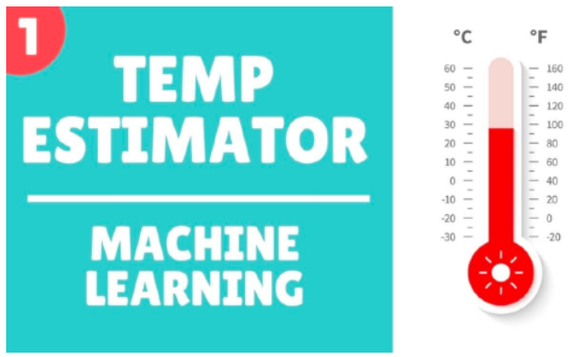
— Tom Mitchell

Learning = Task + Perf. Measure + Data + Algorithm

- The learning task is usually defined by the output of $f : X \to Y$
  - Classification: Y is a finite set, each element is called a class. Historically, binary classification ($Y = \{-1, 1\}$) is the learning task receives most of the studies.

vs.



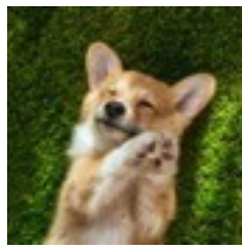


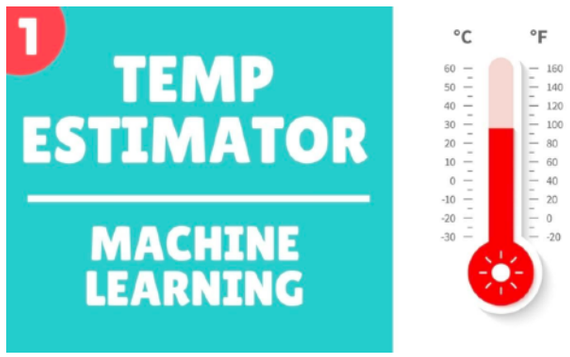
- Regression: Y is real-valued.
- Ranking: Y is the domain of ranking scores.
- Density estimation: Y is the probability.
- …

Learning = Task + Perf. Measure + Data + Algorithm

- The learning task is usually defined by the output of $f : X \to Y$
  - Classification: Y is a finite set, each element is called a class. Historically, binary classification ($Y = \{-1, 1\}$) is the learning task receives most of the studies.

vs.





- Regression: Y is real-valued.
- Ranking: Y is the domain of ranking scores.
- Density estimation: Y is the probability.
- …

Learning = Task + Perf. Measure + Data + Algorithm

- Performance measure defines the best program f^* to find.
- Classification:
  - Accuracy: the probability that the prediction is correct:

$$P(f(x) = y)$$

    - For binary classification:
      - Precision: among all instances that $f(x) = 1$, the proportion that the true class is also $+1$.
      - Recall: among all instances that the true class is $+1$, the proportion that $f(x) = 1$.
- Regression: Mean-square-error $(y - f(x))^2$

Learning = Task + Perf. Measure + Data + Algorithm

- Data is the basic resource for learning f
  - Dataset: $Z = \{z_1, z_2, ..., z_N\}$ instance

The learning scenarios are defined w.r.t. the formulation of data.

- Supervised learning: $z = (x, y)$ labeled instance
  - Each instance consists of both feature and target.
  - Classification, regression, ranking…

- Unsupervised learning (clustering, density estimation…): $z = x$ unlabeled inst.
- Semi-supervised learning: a few labeled and many unlabeled inst.

- Reinforcement learning: Collect data during learning, receive reward instead of the target (action).

Learning = Task + Perf. Measure + Data + Algorithm

- The learning algorithms $\mathcal{A}$ are meta-algorithms with:
  - Input: the set of all $f : \mathcal{F}$, perf. measure $\mathcal{M}$, dataset $\mathcal{S}$
  - Output: the program (model) f
- The process of running the learning algorithm is called training:

$$\mathcal{A} : \mathcal{F} \times \mathcal{M} \times \mathcal{S} \rightarrow f$$

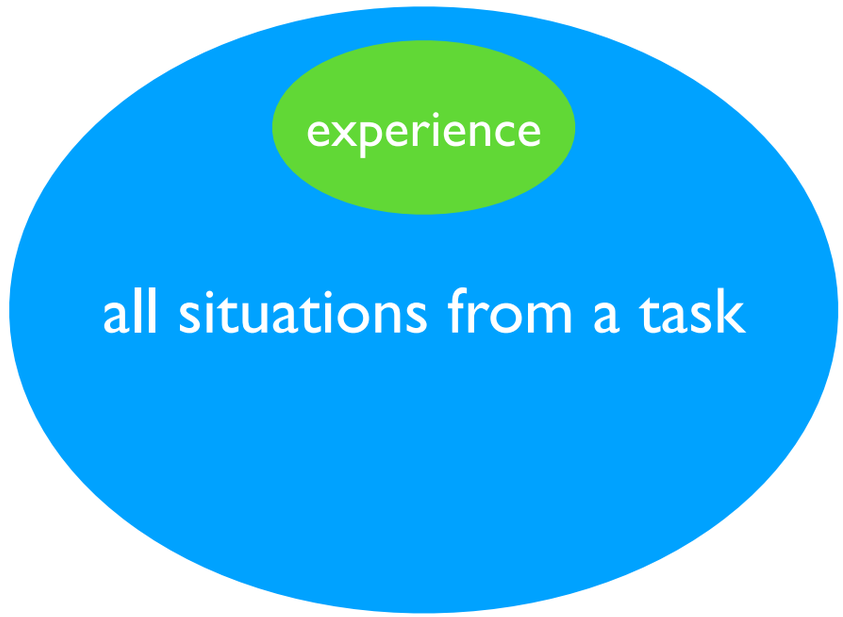
We call $\mathcal{A}$ meta-algorithm since it outputs another program.
Training is usually a process of optimization.

Core of Machine Learning: Generalization

- In machine learning, we assume that all the data are drawn from an unknown distribution $\mathcal{D}$. Both training and testing data are drawn from it.

- Training performance:

$$\hat{R}(f) = \frac{1}{N}\sum_{i=1}^{N} Err(f(x_i), y_i)$$



- Testing performance:

$$R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[Err(f(x), y)]$$

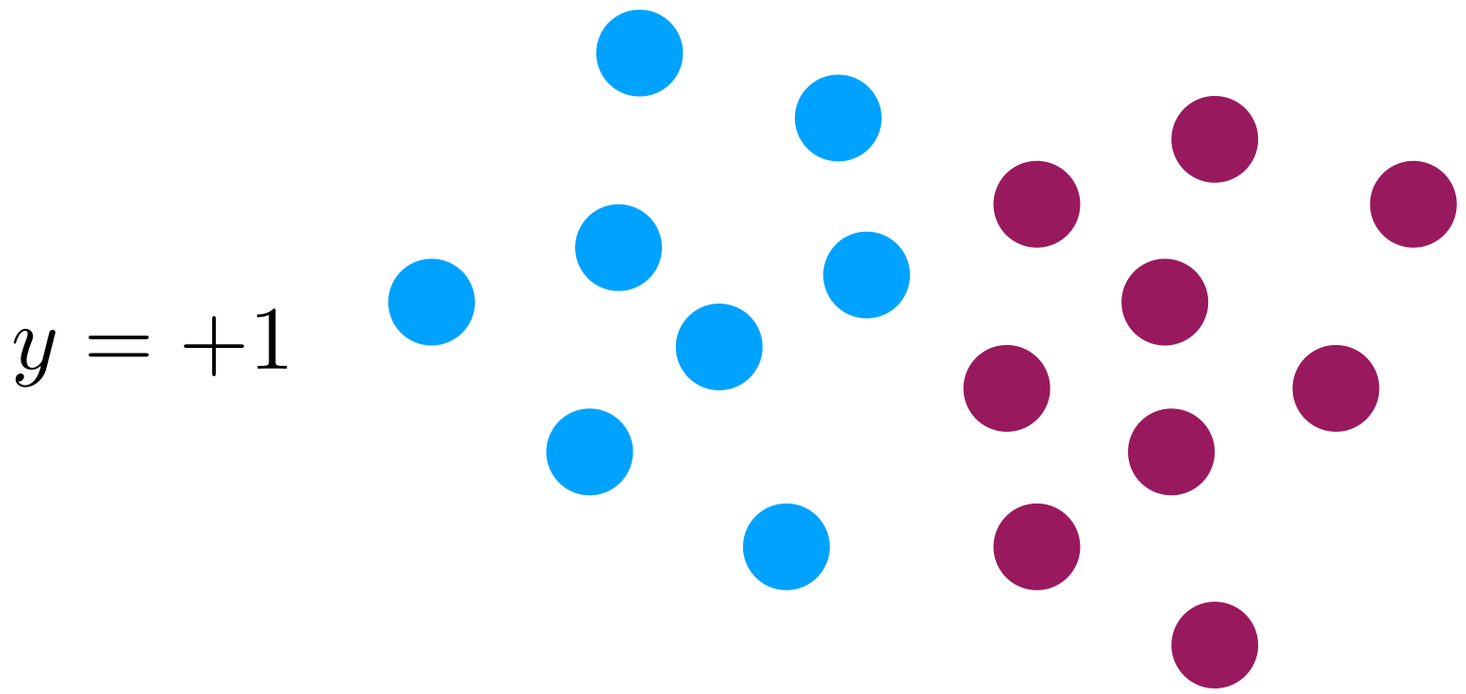
In machine learning, we only observe training data, so we can only control training performance.
But actually we want to control the testing performance!
Generalization is to make training and testing performance close.

Machine Learning: I

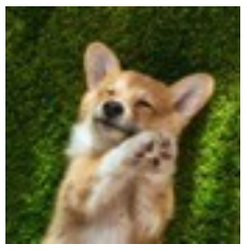
- Basic concepts
- Nearest neighbor classification
- Linear classification
- Bias-Variance Trade-Off & Regularization
- Take-Home Messages

Binary Classification

- Task: learn $f : X \to Y$ with $Y = \{-1, 1\}$
- Performance measure: correctness of classification $\mathbb{I}[f(x) = y]$
- Dataset: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$

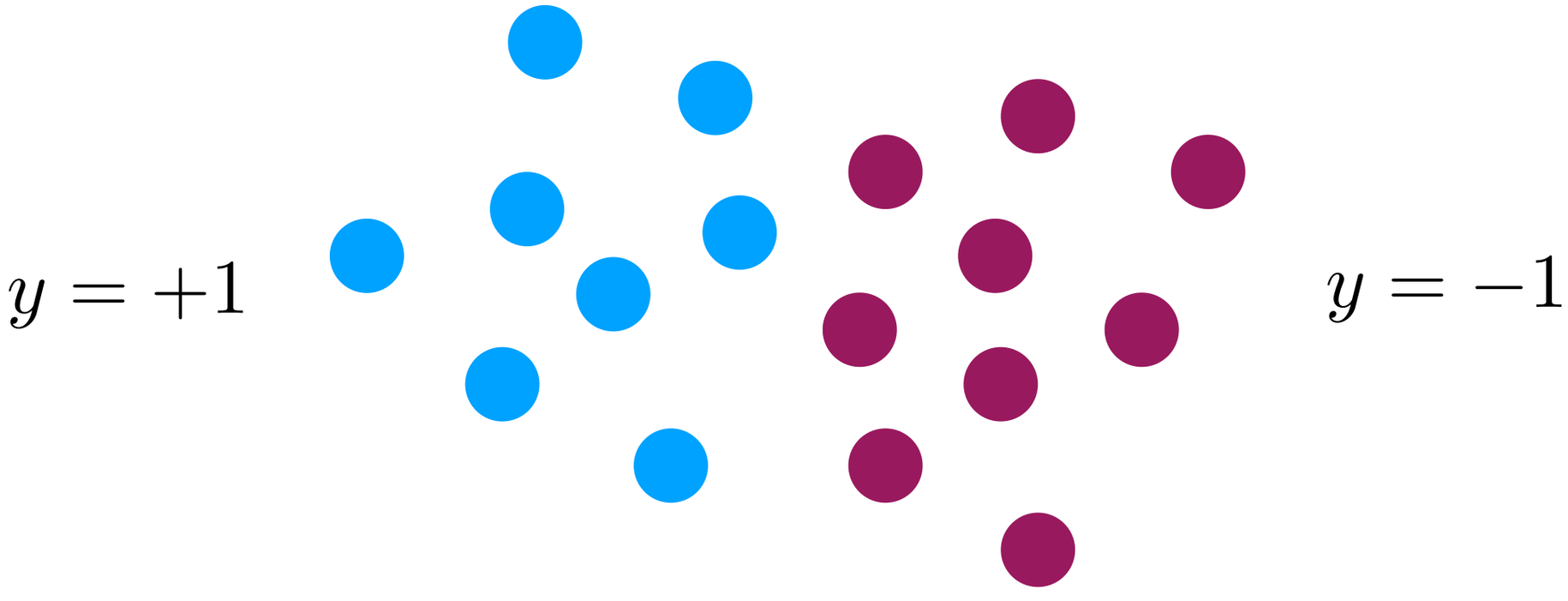


vs.



The Lazy Way of Learning

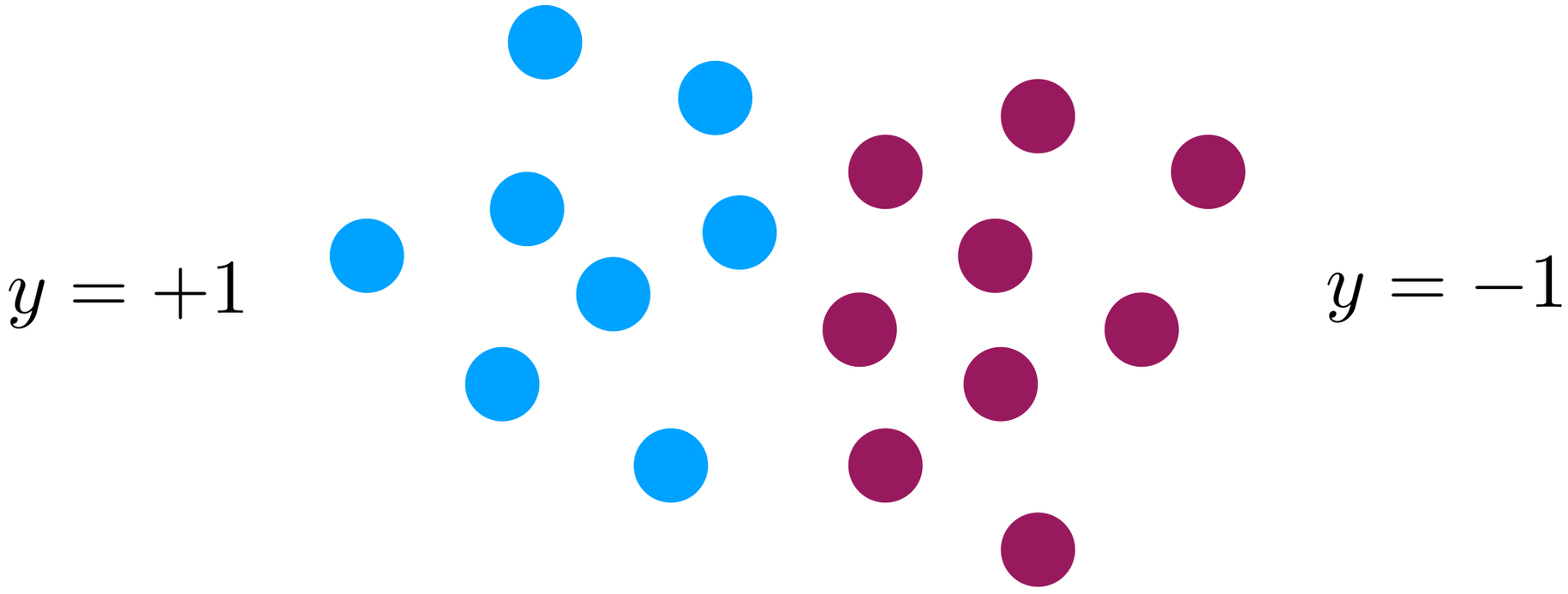
- The learning algorithms $\mathcal{A}$ are meta-algorithms with:
  - Input: the set of all $f : \mathcal{F}$, perf. measure $\mathcal{M}$, dataset $\mathcal{D}$
  - Output: the program (model) f



What is the easiest way of learning?

The Lazy Way of Learning

- The learning algorithms $\mathcal{A}$ are meta-algorithms with:
  - Input: the set of all $f : \mathcal{F}$, perf. measure $\mathcal{M}$, dataset $\mathcal{D}$
  - Output: the program (model) f



What is the easiest way of learning?

Store the dataset, and do 死记硬背.

Nearest Neighbor Classification

- Preparing: Store dataset $\mathcal{D}$, choose distance function $d(x, x')$ and K

- Classification: To classify a new x , examine the K-closest instances in the dataset and assign the most frequently occurring class.

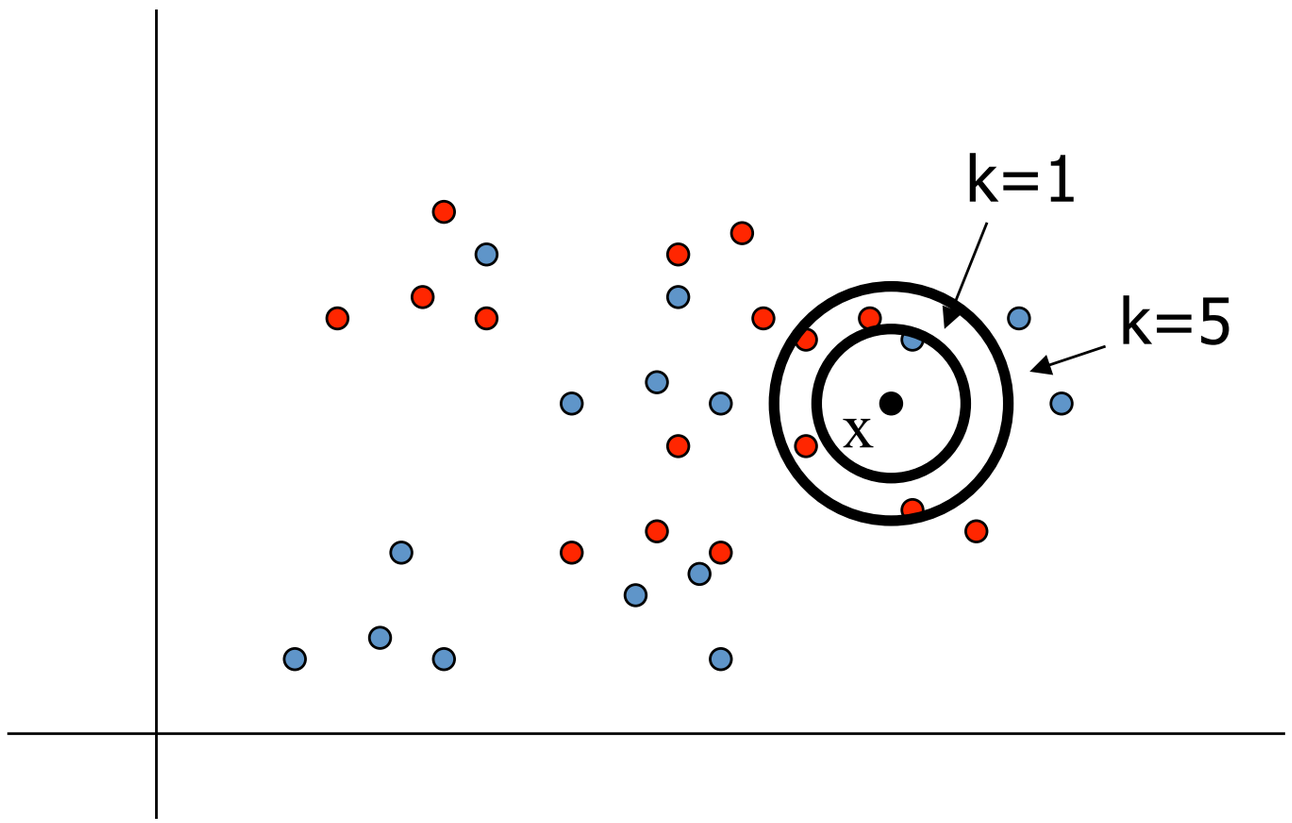


Figure courtesy: David Sontag

Example Results

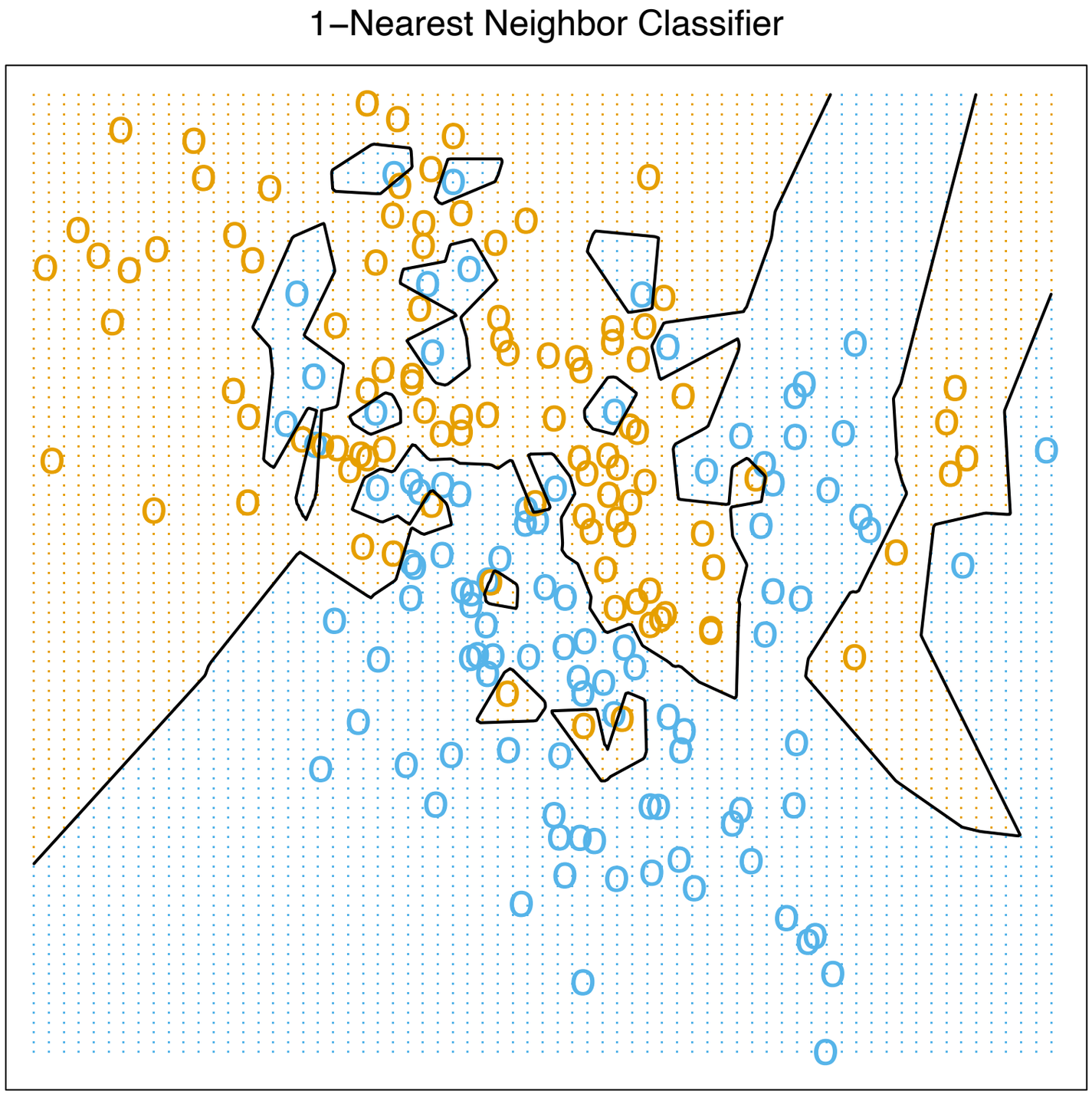


Figure from "Elements of Statistical Learning" book

Example Results

15-Nearest Neighbor Classifier

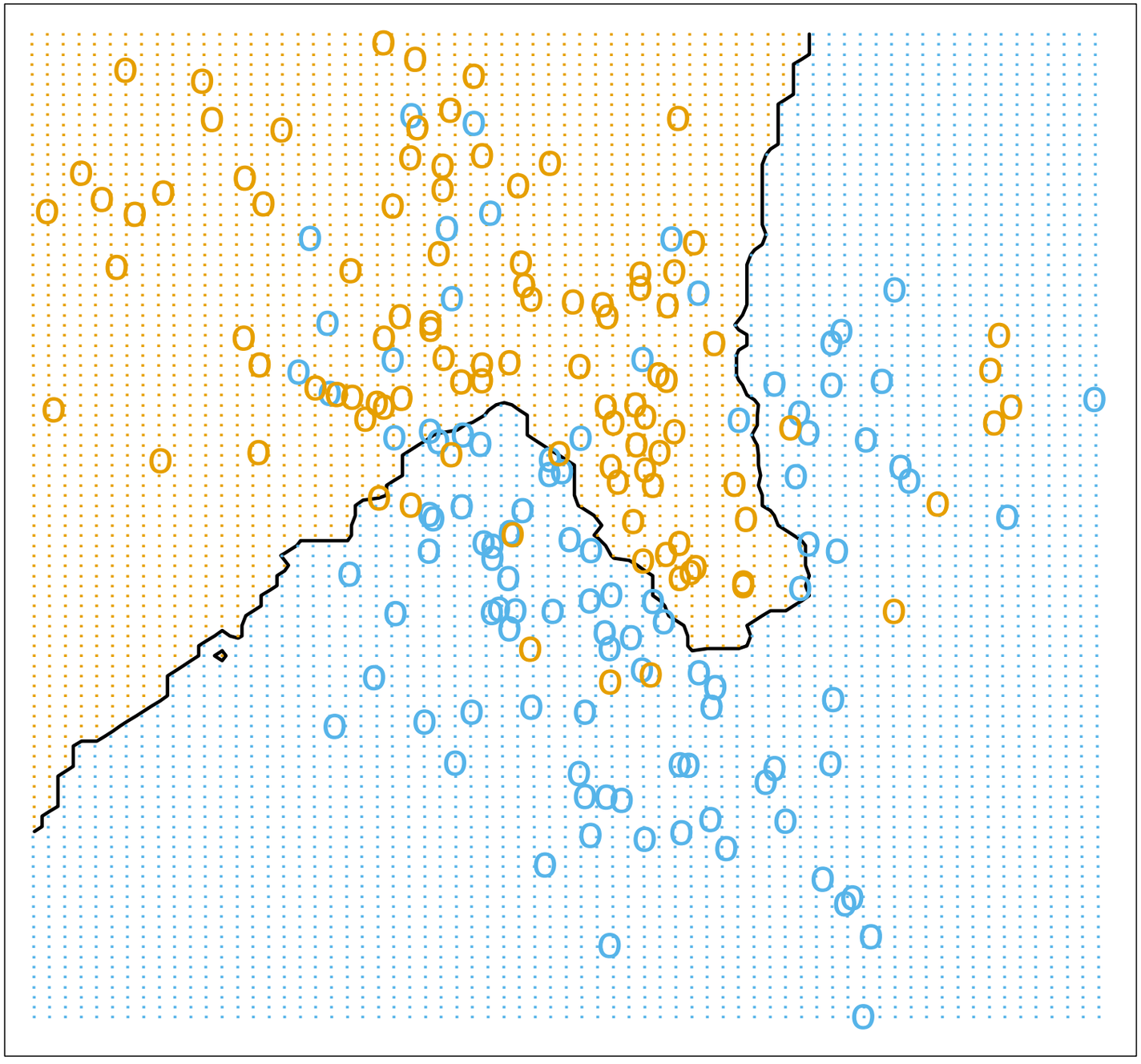
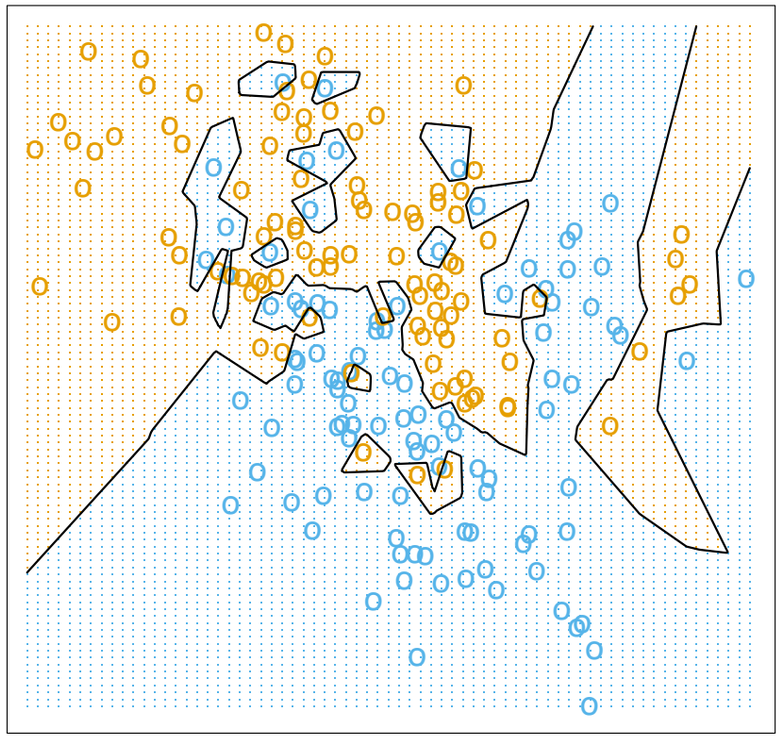


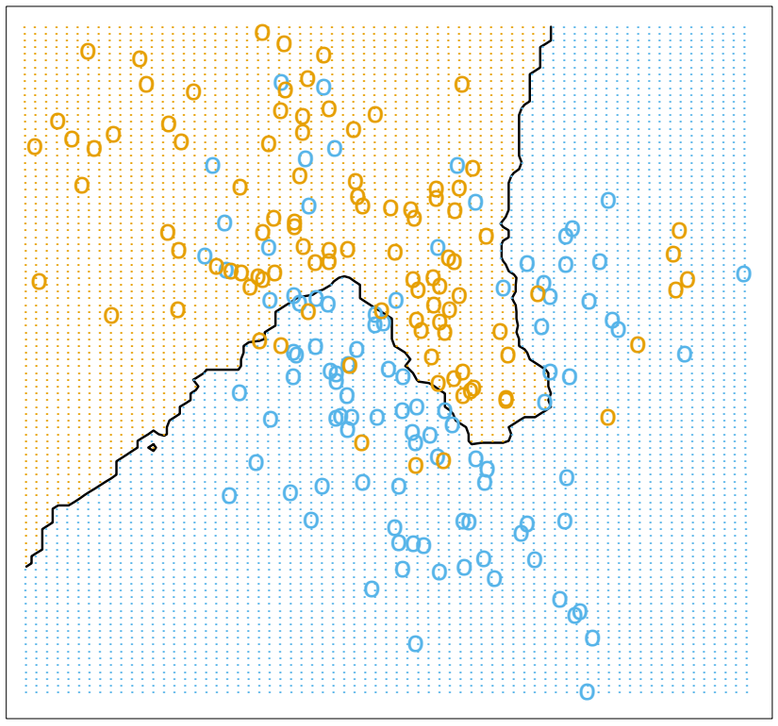
Figure from "Elements of Statistical Learning" book

Example Results

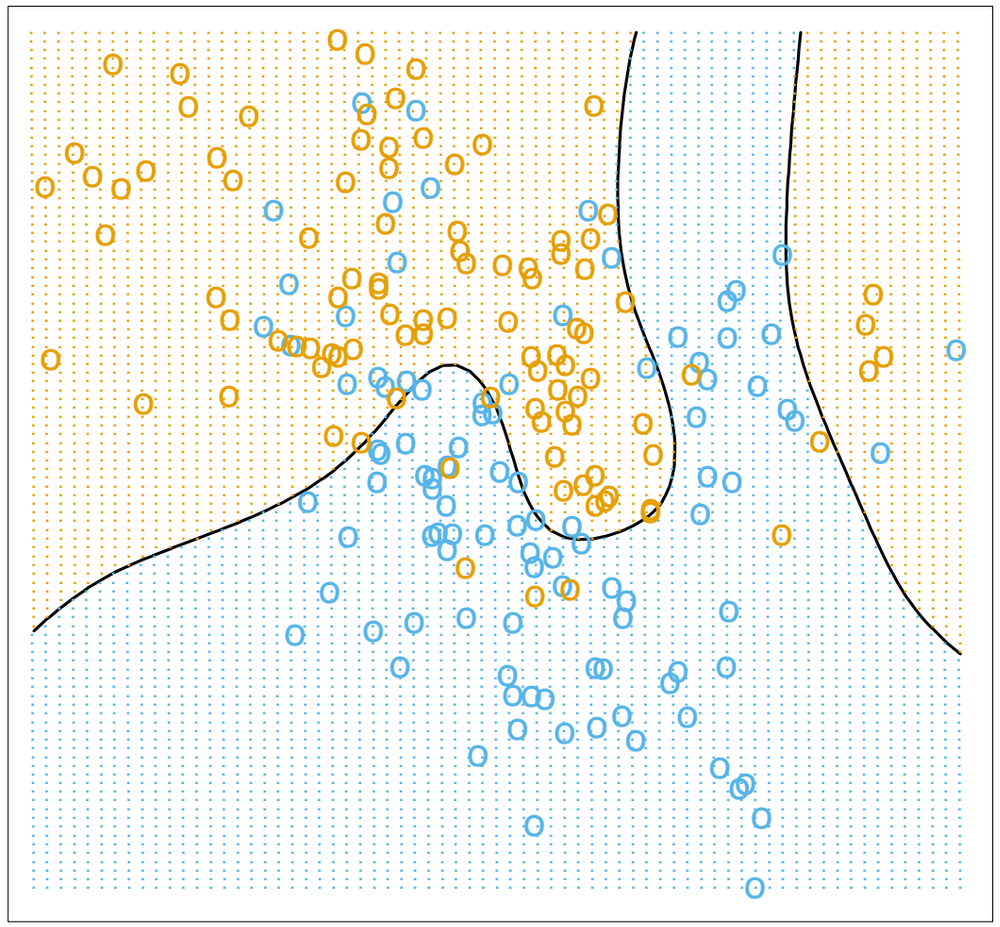
1–Nearest Neighbor Classifier



15-Nearest Neighbor Classifier



Bayes Optimal Classifier



Figures from "Elements of Statistical Learning" book

Choice of Distance & #Neighbor

- The inputs are usually real-valued vectors: $\mathbf{x} = (x_1, x_2, ..., x_d)$
- Euclidean distance: $d(\mathbf{x}, \mathbf{x}') = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2}$
- Cosine distance: $d(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\| \|\mathbf{x}'\|}$
- The choice of K is task-dependent:
  - Small K means more complex decision boundary.
  - Large K means more stable result.

Testing-Time Computation

Nearest neighbor method has no training process,
thus the computation cost of training is very low.
How about testing?

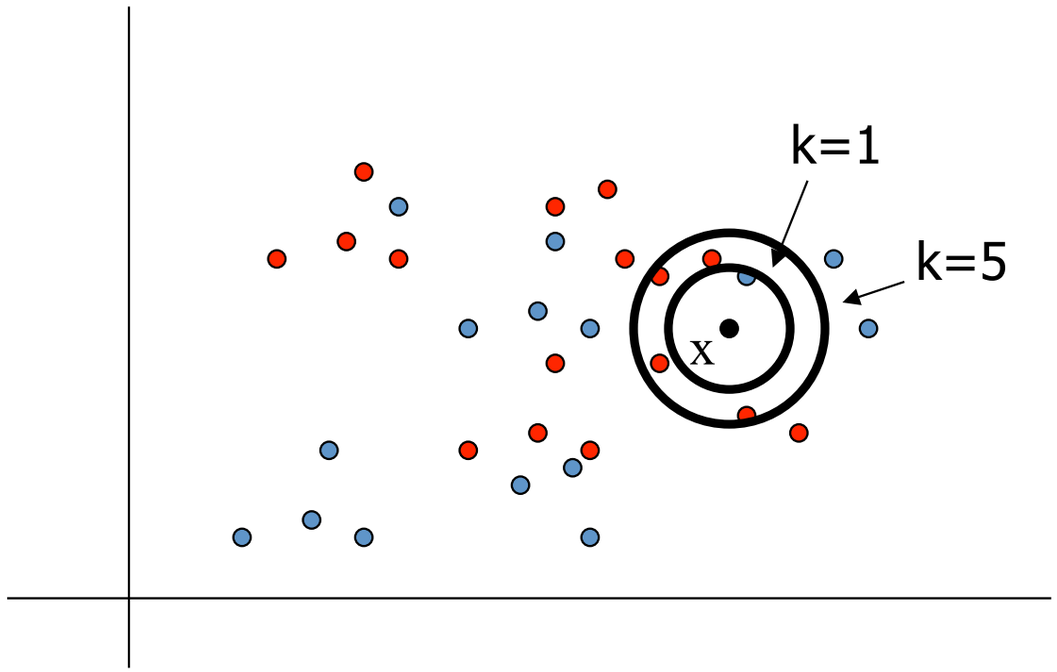
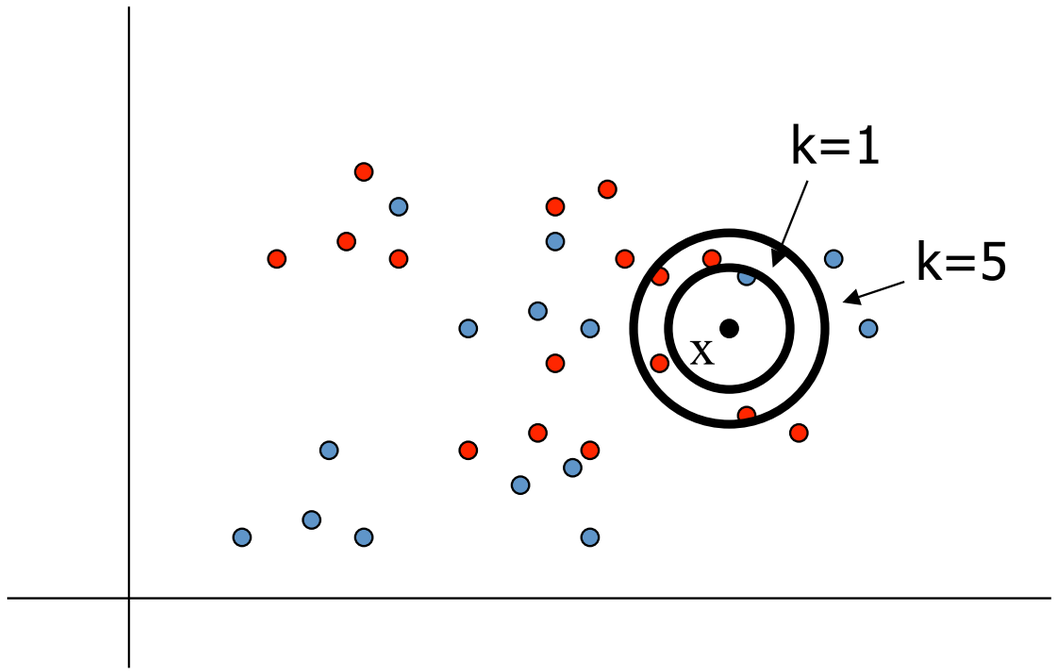


Figure courtesy: David Sontag

Testing-Time Computation

Nearest neighbor method has no training process,
thus the computation cost of training is very low.
How about testing?



Need to find the nearest neighbors!
Trivial implementation costs time linear w.r.t. dataset size!

Figure courtesy: David Sontag

Testing-Time Computation

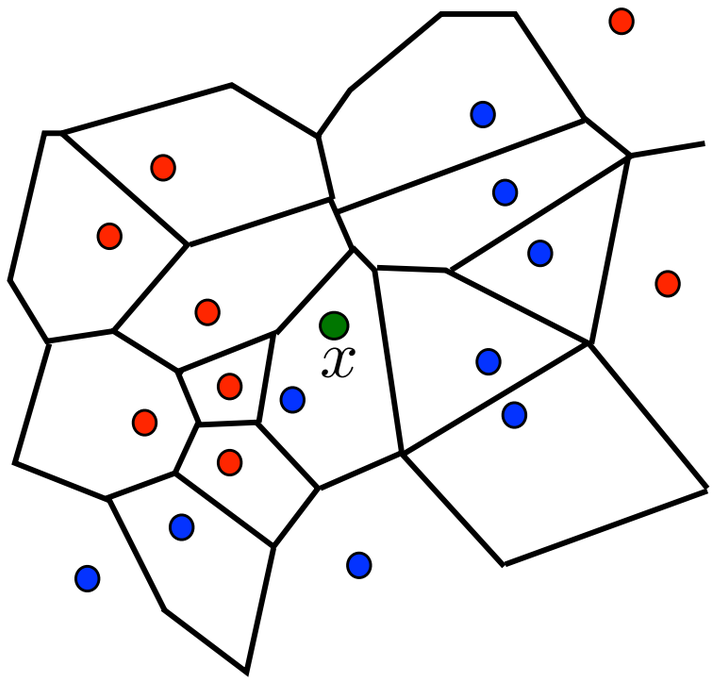
Efficient Indexing: N=2

dimension

- Algorithm:
  - compute Voronoi diagram in $O(m \log m)$.

dataset size

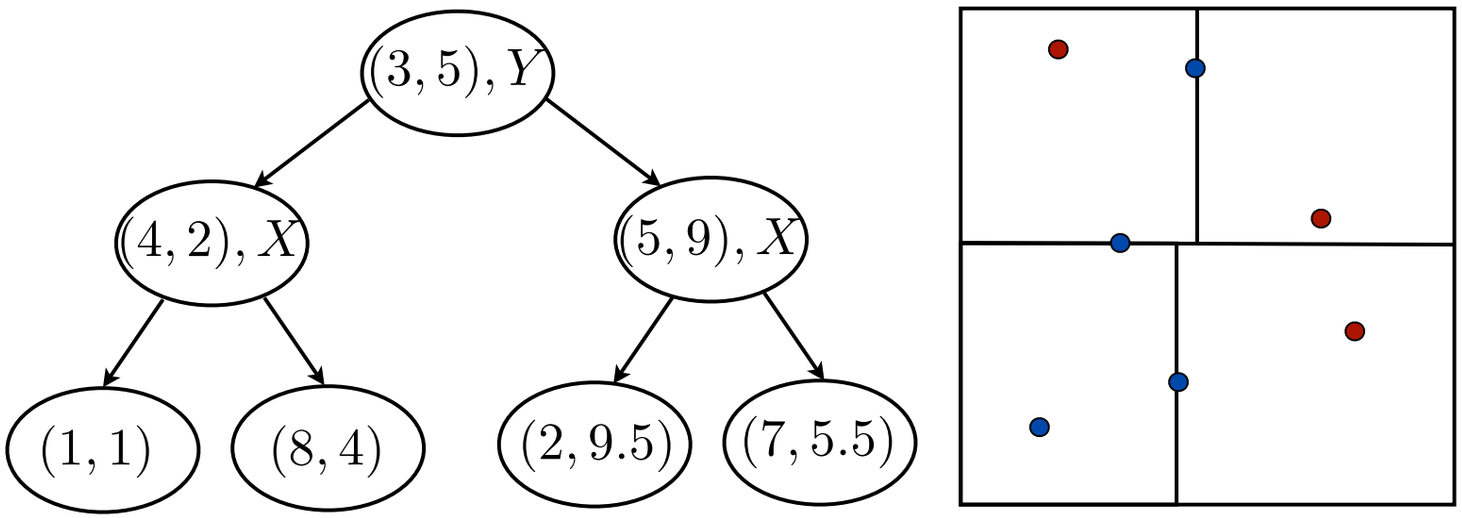
  - point location data structure to determine NN.
  - complexity: $O(m)$ space, $O(\log m)$ time.



Slide Courtesy: Mehyrar Mohri

Testing-Time Computation

Efficient Indexing for N>2: KD trees



Construction algorithm

- Algorithm: for each non-leaf node,
  - choose dimension (e.g., longest of hyperrectangle).
  - choose pivot (median).
  - split node according to (pivot, dimension).

→ balanced tree, binary space partitioning.

Slide Courtesy: Mehyrar Mohri

Curse of Dimensionality

- In theory, we can prove that if the training data is densely distributed in the whole feature space, the error rate of 1-NN will be at most 2 times larger than the optimal classifier.

If 1-NN is already so good, we do we need more complicated learning methods?

Curse of Dimensionality

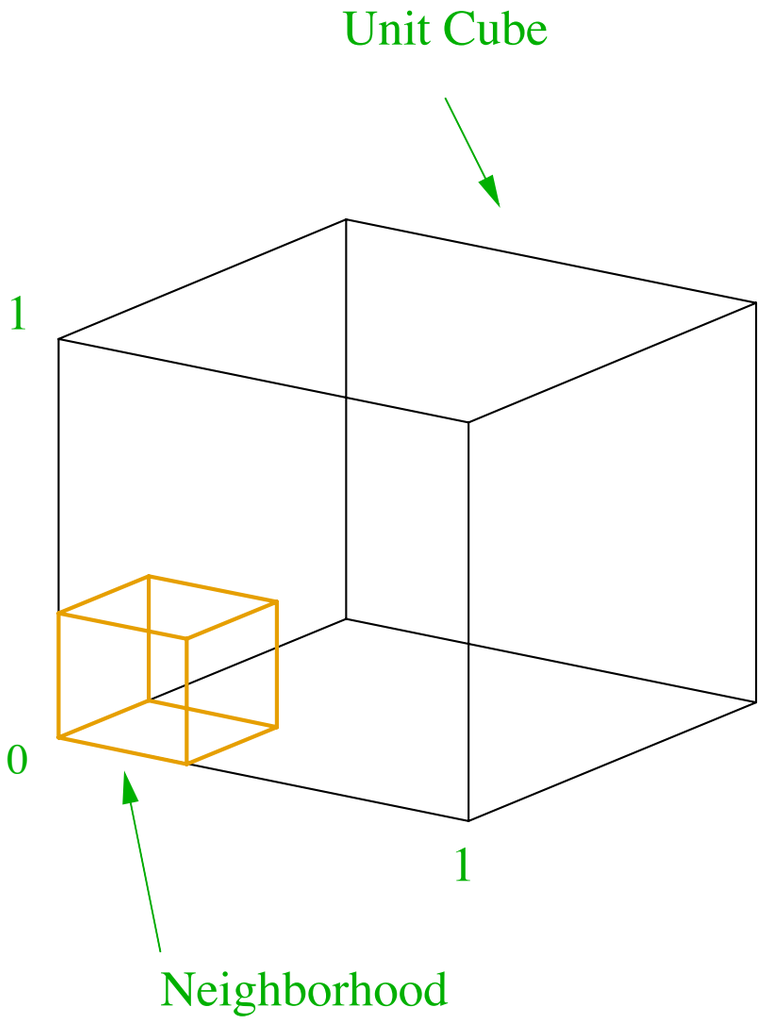
- In theory, we can prove that if the training data is densely distributed in the whole feature space, the error rate of 1-NN will be at most 2 times larger than the optimal classifier.

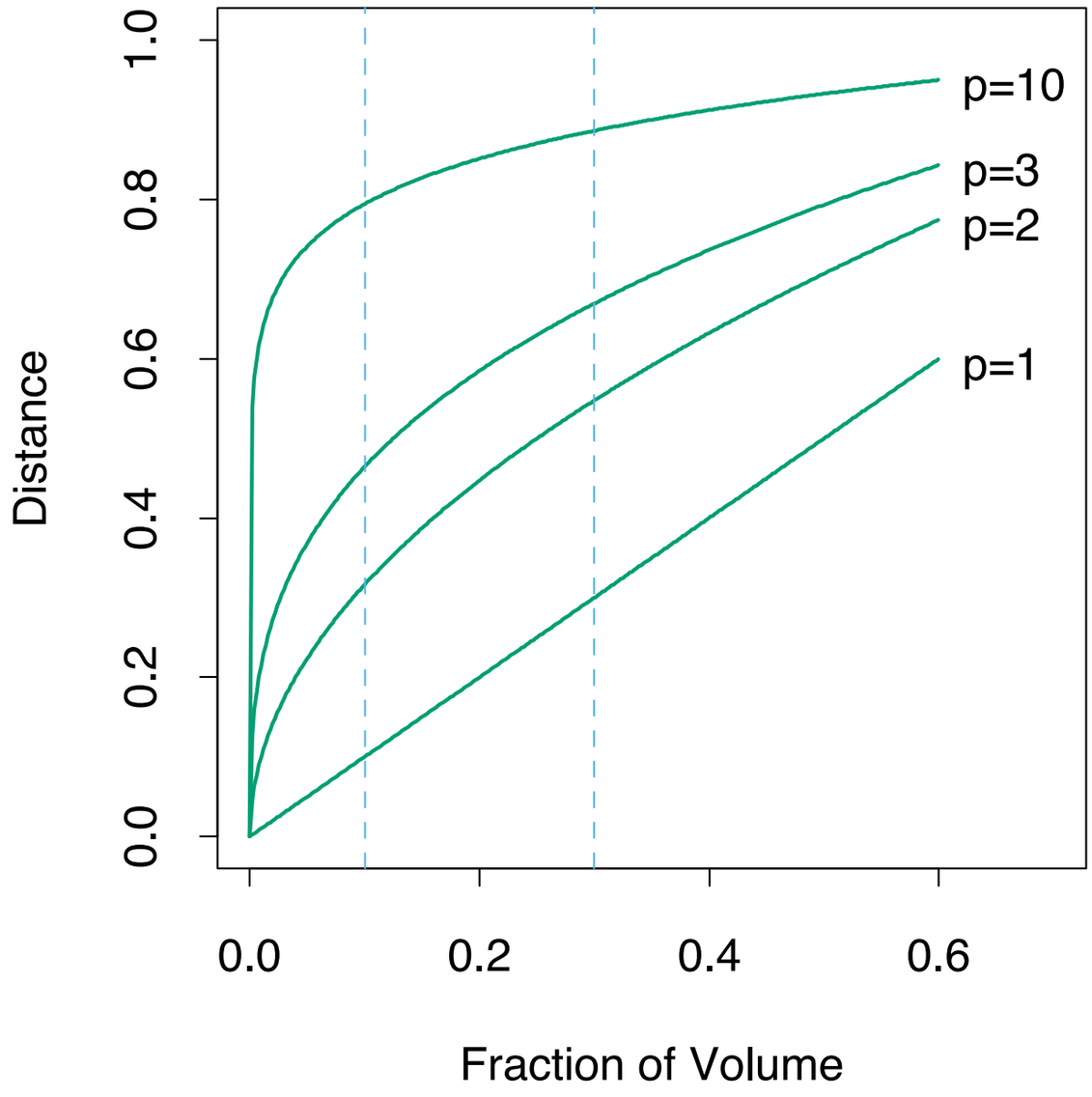
If 1-NN is already so good, we do we need more complicated learning methods?

- The issue is that obtain densely distributed dataset is extremely difficult when the dimension is high.

Consider the 1/2-unit-size ball in d-dim space, if the data is uniformly distributed, then only $1/2^d$ of the data is inside.
Most data in high-dimensional ball locate near the surface.
The cost to obtain dense dataset is exponentially increasing.

Curse of Dimensionality



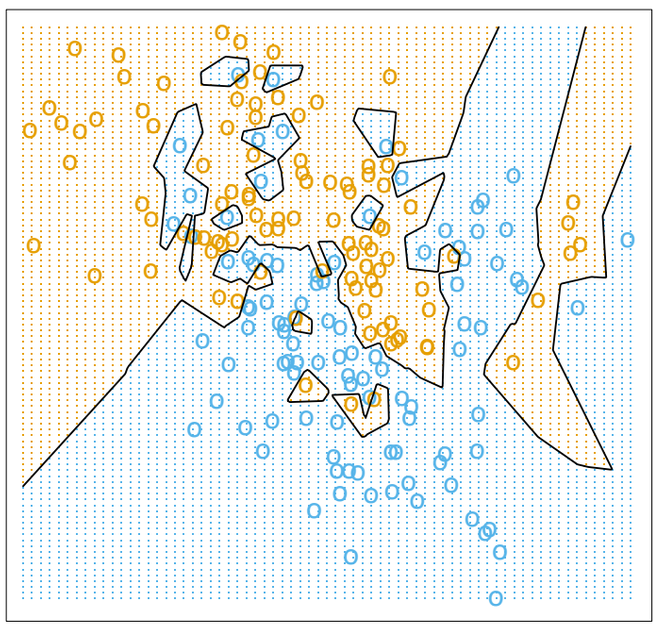


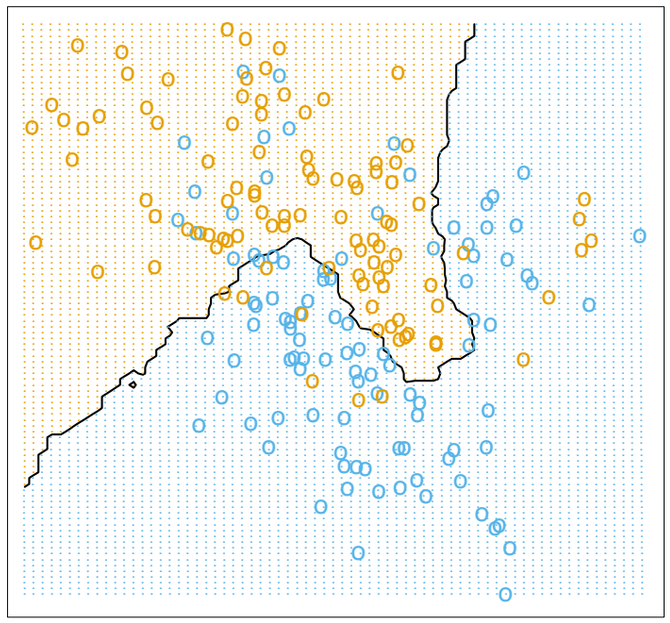
Figures from "Elements of Statistical Learning" book

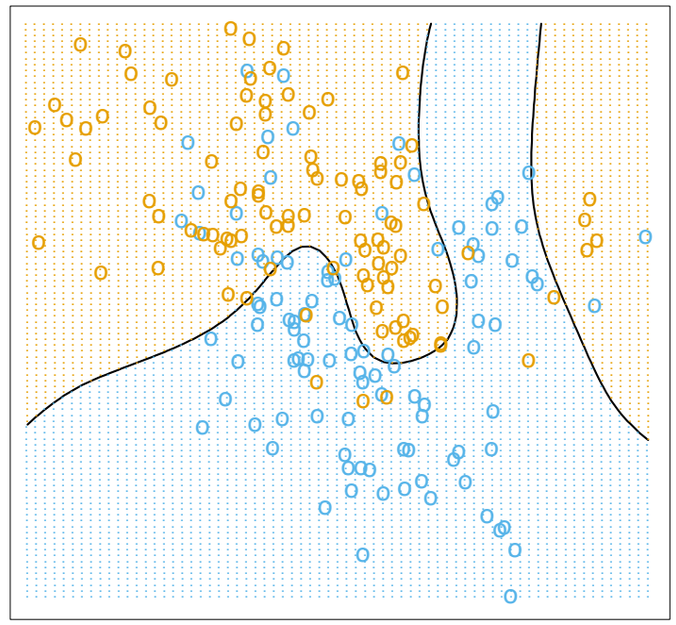
Nearest Neighbor Classification: Pros and Cons

- Pros:
  - Easy to understand and implement.
  - Low (no) training computation cost.
  - Interpretable prediction result.
  - Can approximate any decision function. When the input features are good, can perform surprisingly good, beating more complicated methods.
- Cons:
  - High testing computation cost.
  - Curse of dimensionality, hard to deal with high-dimensional data.

Alternative Approach?







The true decision boundaries usually have structures and regulations.
Use more structured models.

Figures from "Elements of Statistical Learning" book

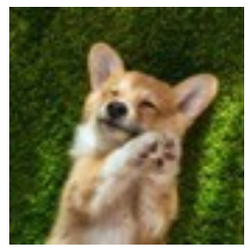
Machine Learning: I

- Basic concepts
- Nearest neighbor classification
- Linear classification
- Bias-Variance Trade-Off & Regularization
- Take-Home Messages

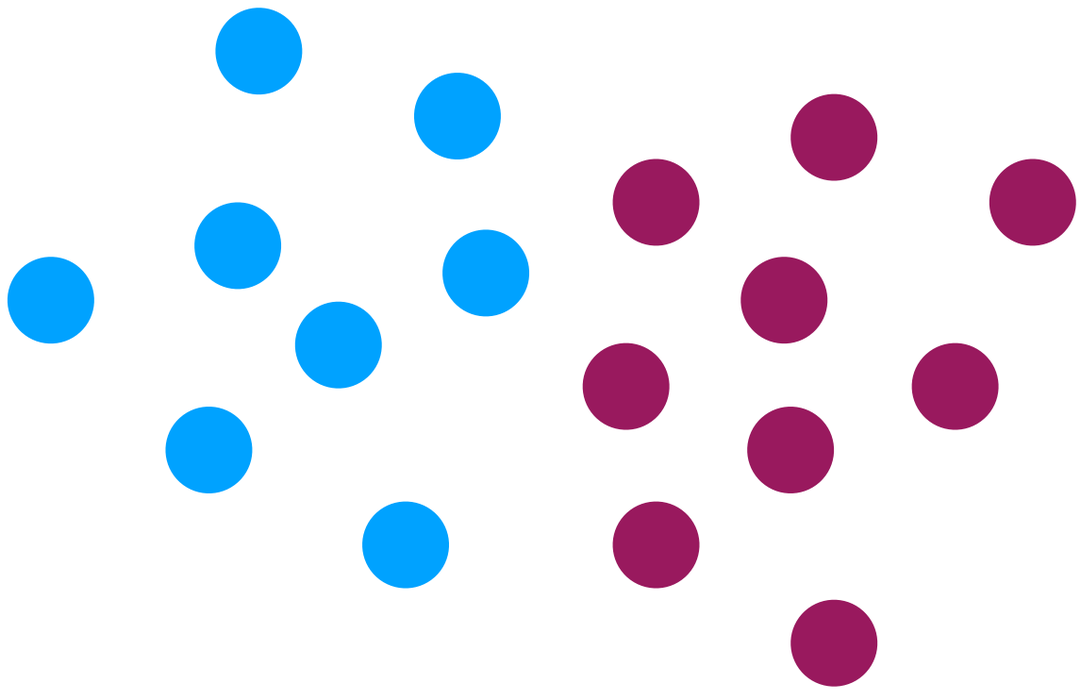
Binary Classification Revisited

- Task: learn $f : X \to Y$ with $Y = \{-1, 1\}$
- Performance measure: correctness of classification $\mathbb{I}[f(x) = y]$
- Dataset: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$

vs.



$y = +1$



$y = -1$

Linear Model & Mean-Squared Error

- Linear model: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{b}$, $\mathbf{w}, \mathbf{x}, \mathbf{b}$ are vectors of dim. d
- Note: equivalent model $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ in dim. $d+1$ since we can then set $\mathbf{x} \to [x_1, x_2, ..., x_d, 1]$ and then hide $\mathbf{b}$.
- Classification rule: $f(x) > 0$ then predict positive, $f(x) < 0$ negative.
- Basic idea: Find the model performs good under the dataset.
- Observation: $\mathbb{I}[f(x) = y]$ is hard to optimize directly.
- Alternative: mean-squared error (MSE): $(f(x) - y)^2$

Least-Square Algorithm

- **Goal**: Find the **best** linear model given the data.
- Many different possible **evaluation** criteria!
- Most common choice is to find the ***w*** that minimizes:

$$Err(\mathbf{w}) = \sum_{i=1:n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

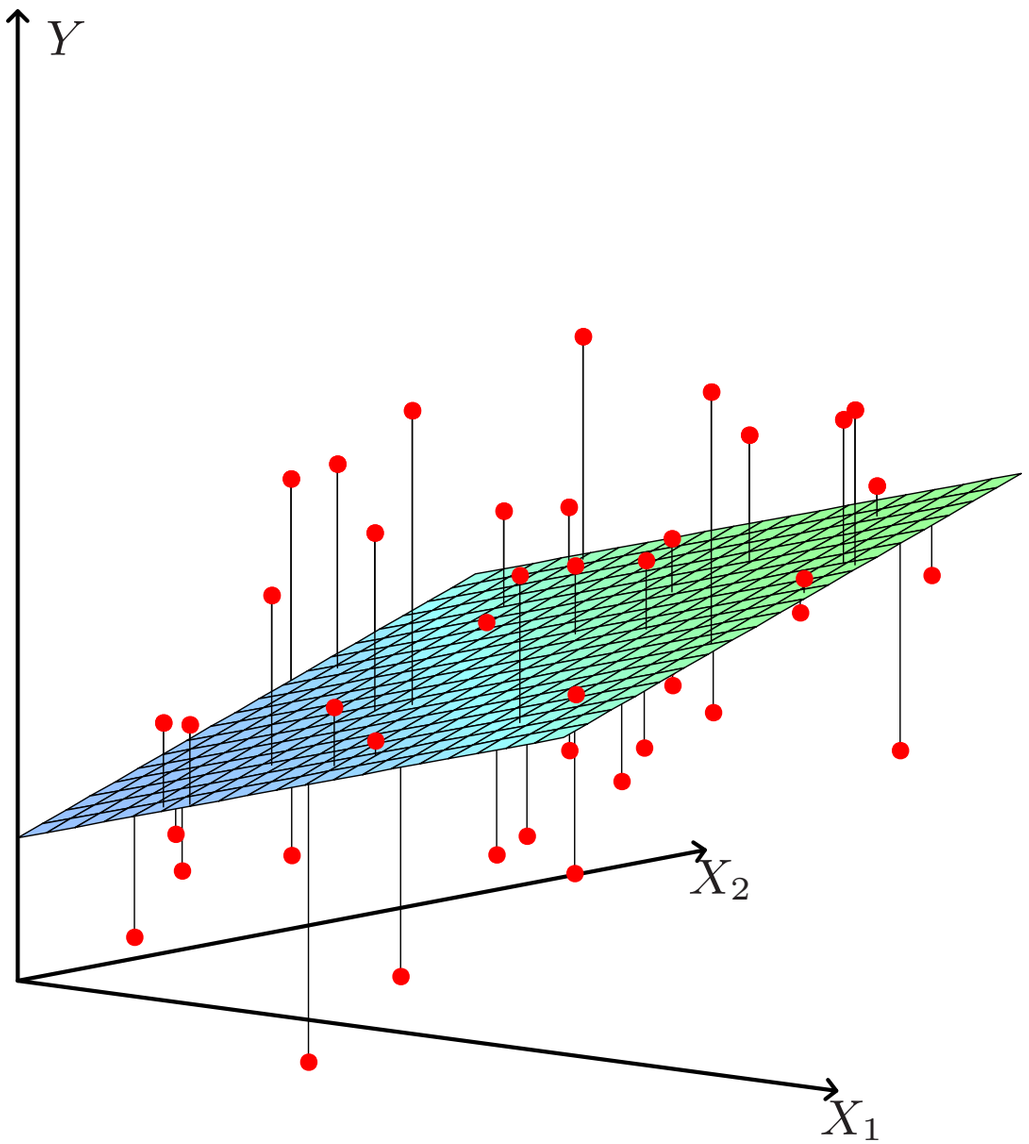
(A note on notation: Here ***w*** and ***x*** are column vectors of size *m+1*.)

data dim.

Find the model with minimum MSE on the dataset.
Actually the linear regression method.

Slide Courtesy: Joelle Pineau

Least-Square Algorithm



Regression example.

Figure from "Elements of Statistical Learning" book

Optimization: Analytical Solution

- Re-write in matrix notation: $f_{\boldsymbol{w}}(X) = X\boldsymbol{w}$

$$Err(\boldsymbol{w}) = (Y - X\boldsymbol{w})^T (Y - X\boldsymbol{w})$$

where X is the $n \times m$ matrix of input data,
Y is the $n \times 1$ vector of output data,
$\boldsymbol{w}$ is the $m \times 1$ vector of weights.

- To minimize, take the derivative w.r.t. $\boldsymbol{w}$:

$$\partial Err(\boldsymbol{w})/\partial \boldsymbol{w} = -2\, X^T (Y - X\boldsymbol{w})$$

  - You get a system of m equations with m unknowns.
- Set these equations to 0: $X^T (Y - X\boldsymbol{w}) = 0$
  - *Remember that derivative has to be 0 at a minimum of Err(w)*

Slide Courtesy: Joelle Pineau

Optimization: Analytical Solution

- We want to solve for $\boldsymbol{w}$: $X^T (Y - X\boldsymbol{w}) = 0$

- Try a little algebra: $X^T Y = X^T X \boldsymbol{w}$

$\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T Y$

($\hat{\boldsymbol{w}}$ denotes the estimated weights)

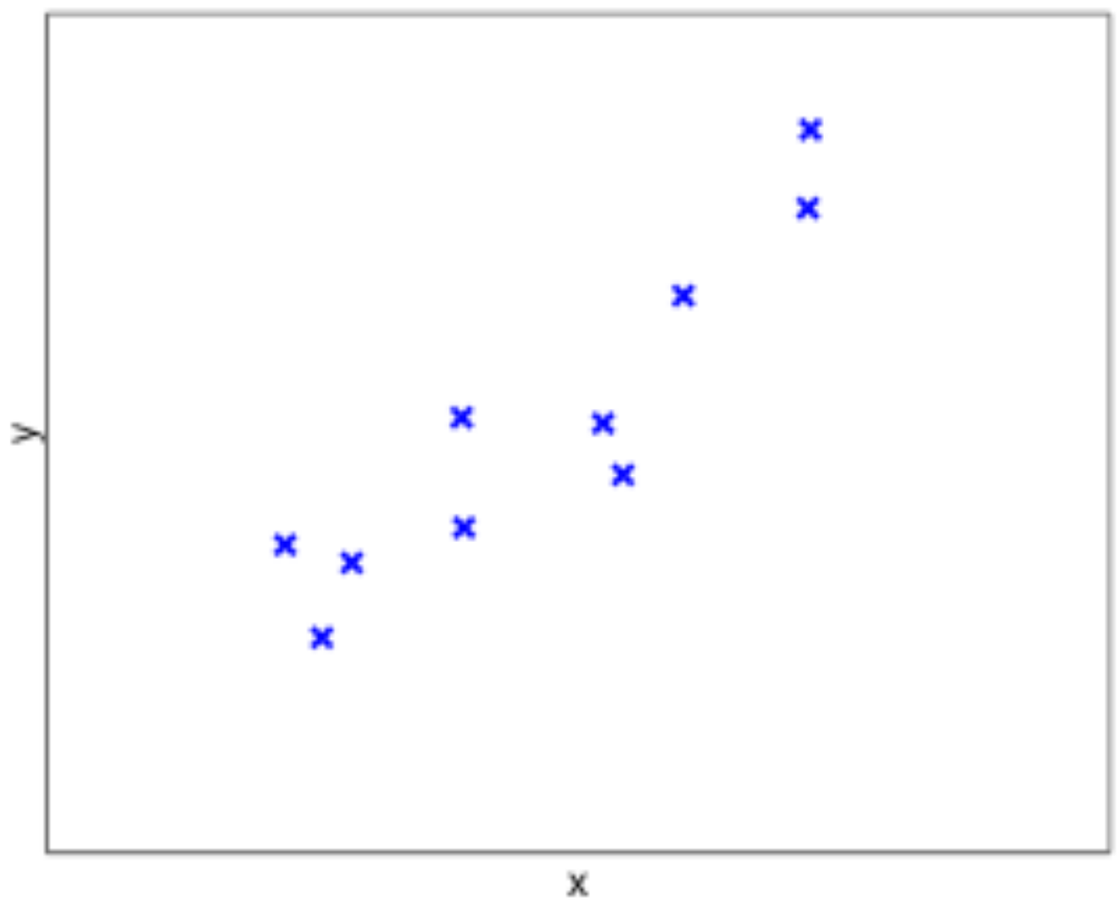
- Train set predictions: $\hat{Y} = X\hat{\boldsymbol{w}} = X (X^T X)^{-1} X^T Y$

- Predict new data $X' \rightarrow Y'$: $Y' = X'\hat{\boldsymbol{w}} = X' (X^T X)^{-1} X^T Y$

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Slide Courtesy: Joelle Pineau

Example



x	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.10	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

What is a plausible estimate of ***w*** ?

Try it!

Example

$$X^T X =$$

$$\begin{bmatrix} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0.86 & 1 \\ 0.09 & 1 \\ -0.85 & 1 \\ 0.87 & 1 \\ -0.44 & 1 \\ -0.43 & 1 \\ -1.10 & 1 \\ 0.40 & 1 \\ -0.96 & 1 \\ 0.17 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}$$

Slide Courtesy: Joelle Pineau

Example

$$X^T Y =$$

$$\begin{bmatrix} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

$$= \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix}$$

Slide Courtesy: Joelle Pineau

Example

$$X^T Y = \begin{bmatrix} 0.86 & 0.09 & -0.85 & 0.87 & -0.44 & -0.43 & -1.10 & 0.40 & -0.96 & 0.17 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 2.49 \\ 0.83 \\ -0.25 \\ 3.10 \\ 0.87 \\ 0.02 \\ -0.12 \\ 1.81 \\ -0.83 \\ 0.43 \end{bmatrix}$$

$$= \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix}$$

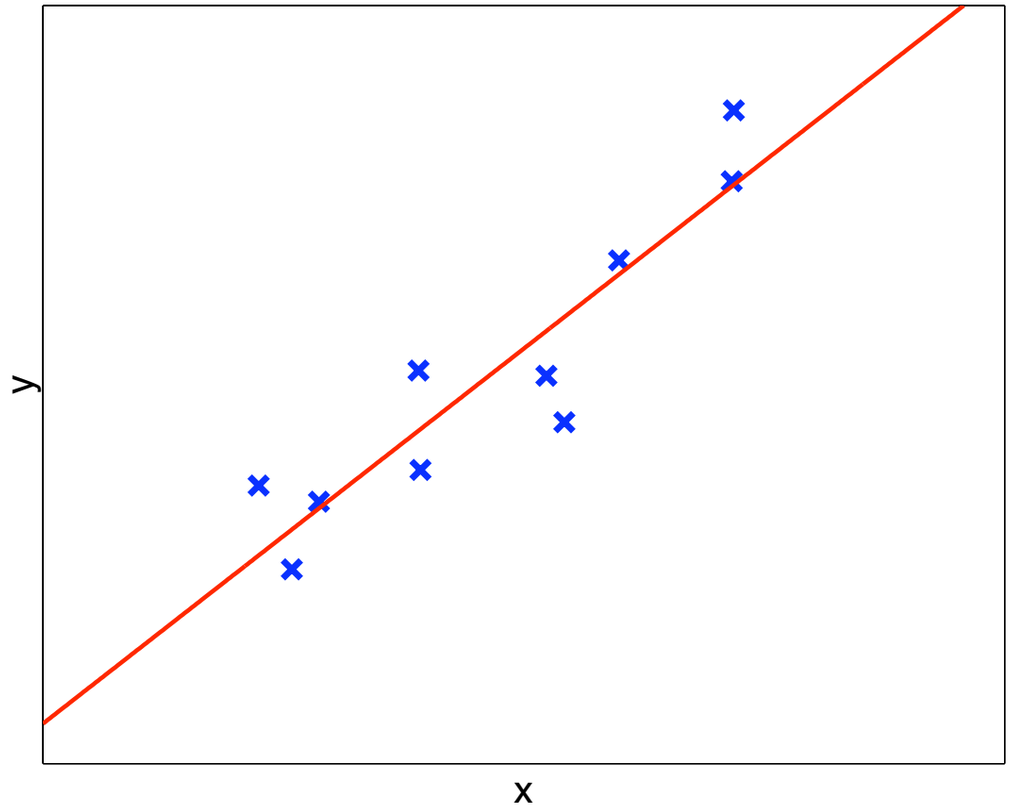
For classification, these are -1 or +1.

Slide Courtesy: Joelle Pineau

Example

$$\mathbf{w} = (X^T X)^{-1} X^T Y = \begin{bmatrix} 4.95 & -1.39 \\ -1.39 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 6.49 \\ 8.34 \end{bmatrix} = \begin{bmatrix} 1.60 \\ 1.05 \end{bmatrix}$$

So the best fit line is $y = 1.60x + 1.05$.



Slide Courtesy: Joelle Pineau

Computation Cost

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

- What operations are necessary?
  - Overall: 1 matrix inversion + 3 matrix multiplications
  - X^TX (other matrix multiplications require fewer operations.)
    - X^T is mxn and X is nxm, so we need nm^2 operations.
  - $(X^TX)^{-1}$
    - X^TX is mxm, so we need m^3 operations.

- We can do linear regression in polynomial time, but handling large datasets (many examples, many features) can be problematic.

Slide Courtesy: Joelle Pineau

Optimization: Gradient Descent

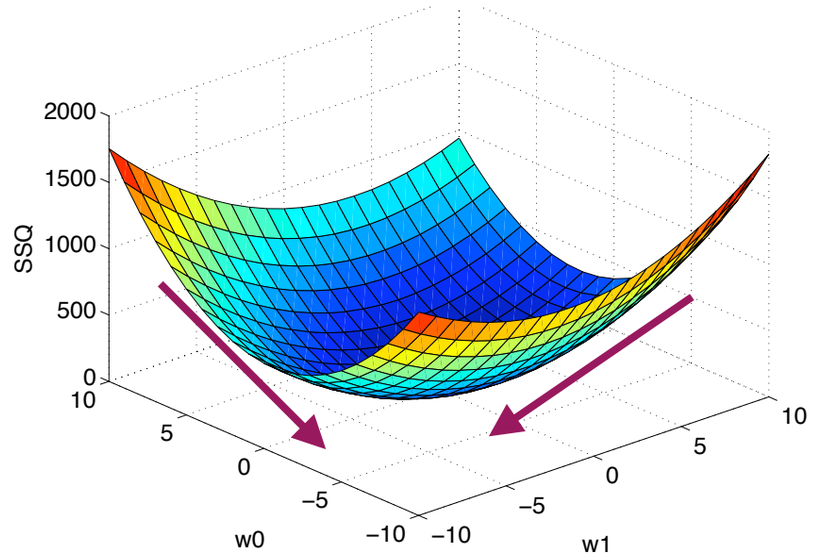
- MSE: $(\mathbf{w}^T\mathbf{x} - y)^2$ is a convex function w.r.t. $\mathbf{w}$

Given an initial weight vector $\mathbf{w}_0$,

Do for $k=1, 2, \ldots$

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \, \partial Err(\mathbf{w}_k)/\partial \mathbf{w}_k$$

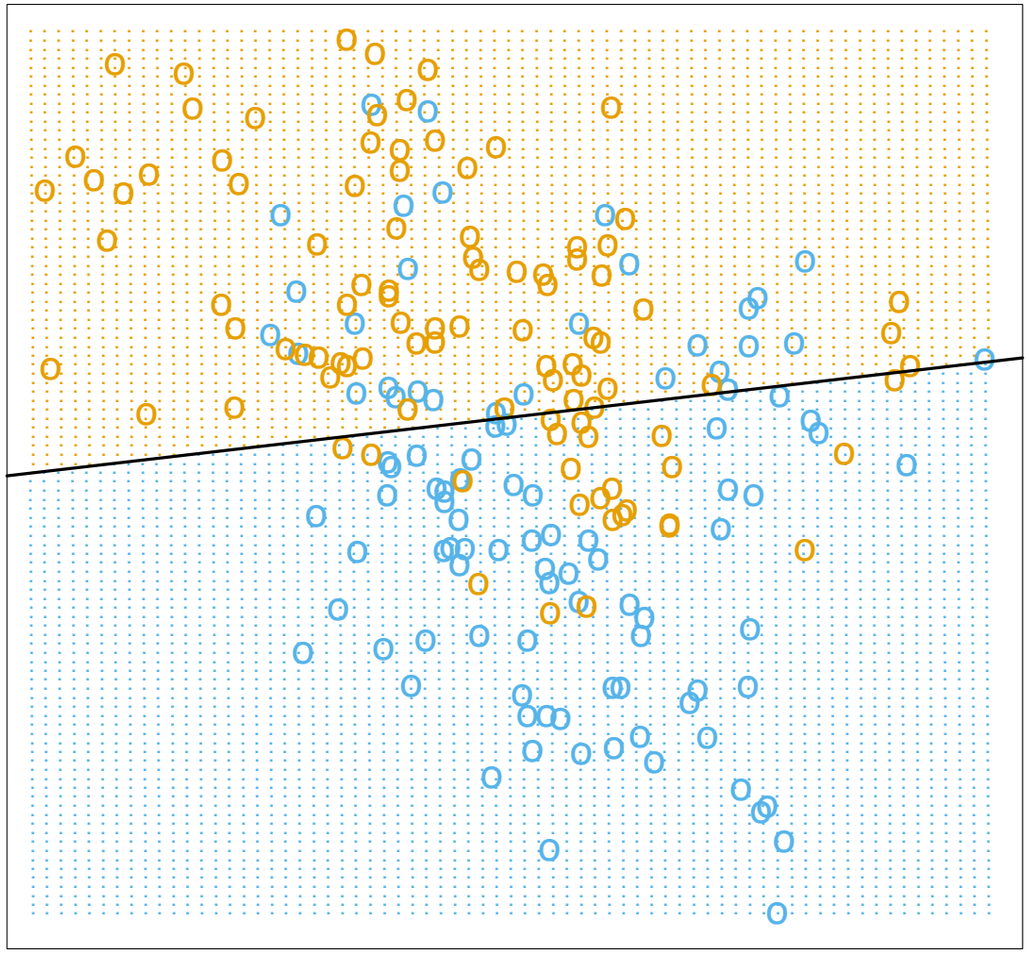
End when $|\mathbf{w}_{k+1} - \mathbf{w}_k| < \varepsilon$



Tracking the gradient direction to find the minimum

Gradient descent is guaranteed to find the global minimum for convex functions.
But the solution is unique only when $N \geq d$

Decision Boundary



To approximate non-linear decision boundary, we can augment feature space:

$$[x_1, x_2, ..., x_d] \rightarrow [x_1, x_2, ..., x_d, x_1^2, x_2^2, ..., x_d^2, ...]$$

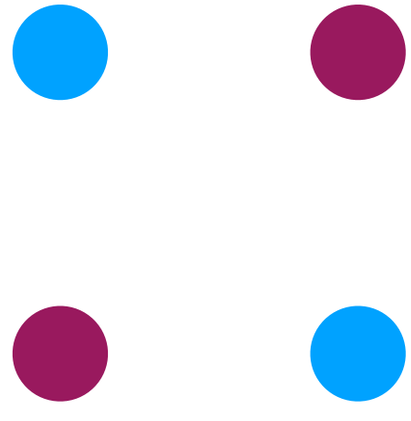
and then set $\mathbf{w}$ in the new high-dimensional space.

Linear Classification with MSE: Pros and Cons

- Pros:
  - Easy to understand and implement.
  - Reasonable training and testing computation cost.
  - Interpretable prediction result:
    - Need a few data points in the high dimension.

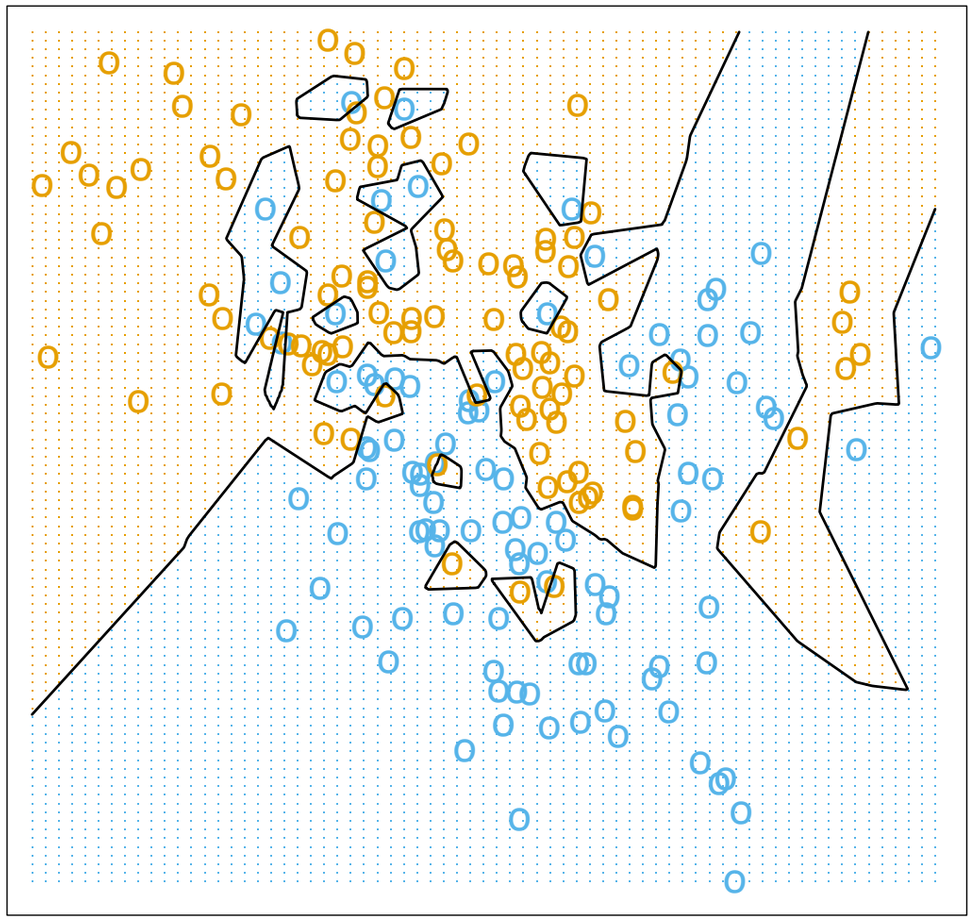
Each dim. of $\mathbf{w}$ is the importance of the feature

- Cons:
  - Can not represent complex function.

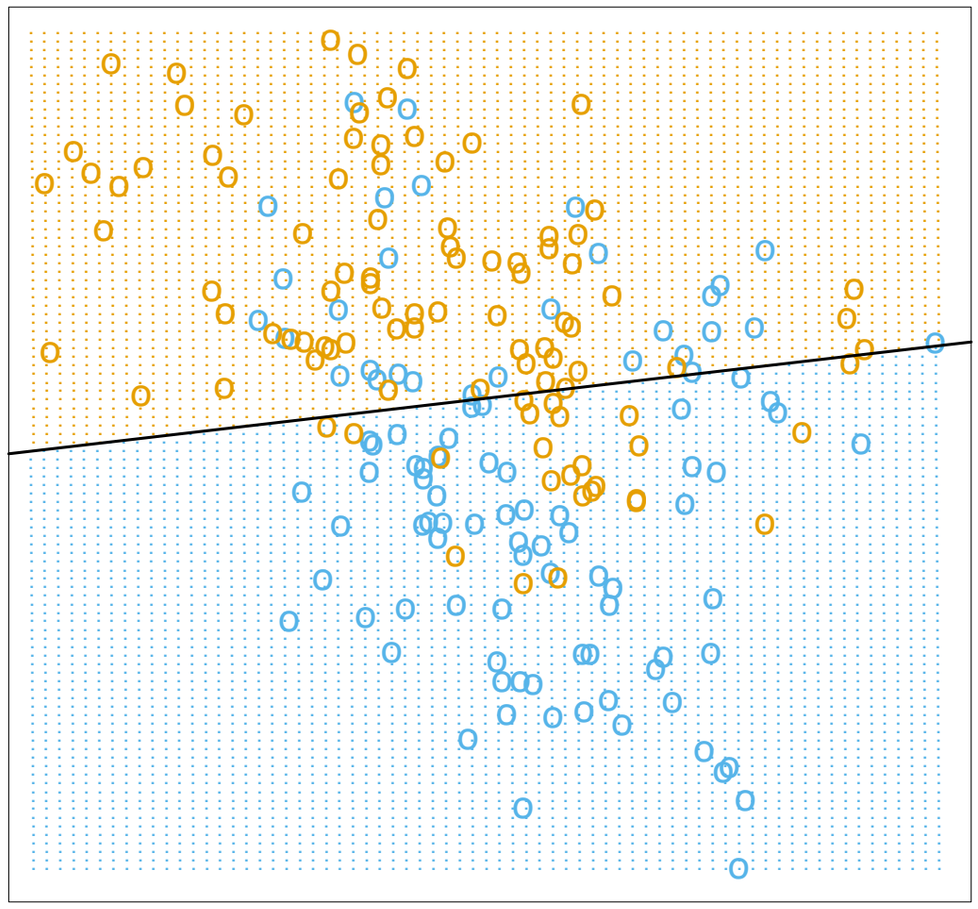


Nearest Neighbor vs. Linear Model

1–Nearest Neighbor Classifier



Linear Regression of 0/1 Response



All learning models are in between these two extremes:
balancing flexibility and stability.
Can we find a general way to achieve this balance?

Machine Learning: I

- Basic concepts
- Nearest neighbor classification
- Linear classification
- Bias-Variance Trade-Off & Regularization
- Take-Home Messages

Generalization Revisited

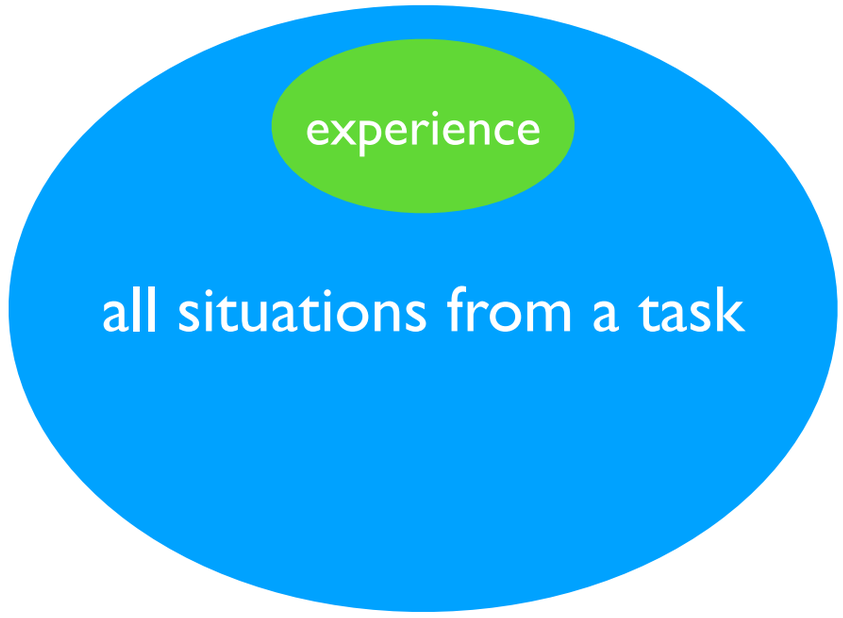
- In machine learning, we assume that all the data are drawn from an unknown distribution $\mathcal{D}$. Both training and testing data are drawn from it.

- Training performance:

$$\hat{R}(f) = \frac{1}{N}\sum_{i=1}^{N} Err(f(x_i), y_i)$$

- Testing performance:

$$R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[Err(f(x), y)]$$



In linear regression, we minimize the MSE on the training data.
What is the true testing performance?

Bias & Variance Trade-Off

- For linear regression, the expected testing performance is:

$$\mathbb{E}_{\mathcal{S}\sim\mathcal{D}^N}[R(\mathbf{w})] = \mathbb{E}_{(x,y)\sim\mathcal{D},\mathcal{S}\sim\mathcal{D}^N}[(\mathbf{w}^T\mathbf{x} - y)^2]$$

- Bias-Variance decomposition: $\mathbb{E}_{\mathcal{S}\sim\mathcal{D}^N}[R(\mathbf{w})]$ can be decomposed as

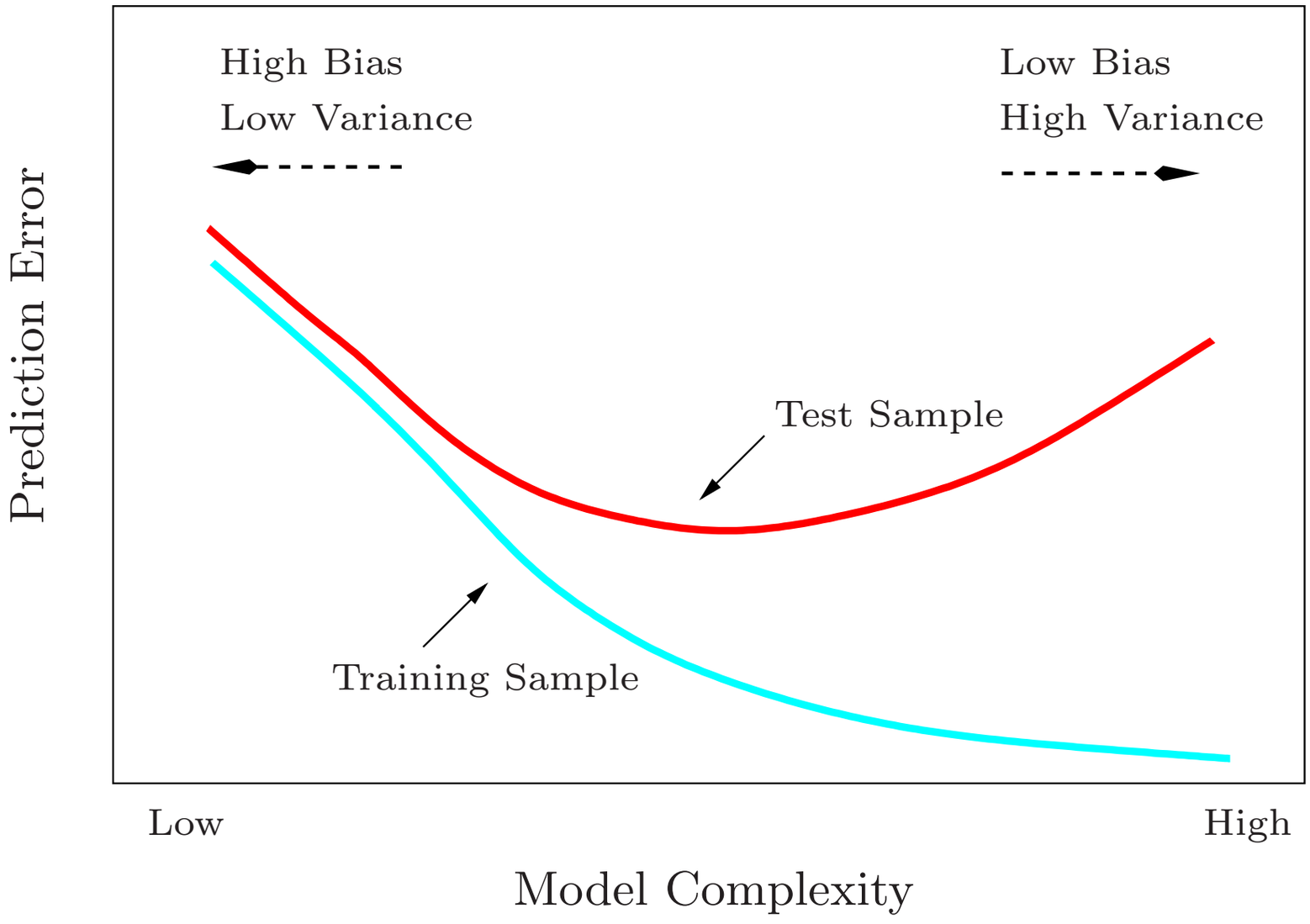
$$\underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathbb{E}_{\mathcal{S}\sim\mathcal{D}^N}[(\mathbf{w}^T\mathbf{x} - \mathbb{E}_{\mathcal{S}\sim\mathcal{D}^N}[\mathbf{w}^T\mathbf{x}])^2]\right]}_{\text{variance: deviation from the mean}} + \underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[(\mathbb{E}_{\mathcal{S}\sim\mathcal{D}^N}[\mathbf{w}^T\mathbf{x}] - y)^2\right]}_{\text{bias: error of the mean to the true target}}$$

Model with high variance: The result is not stable, e.g. 1-nearest neighbor.
Model with high bias: The model is not flexible enough, e.g. linear regression.

Bias & Variance Trade-Off

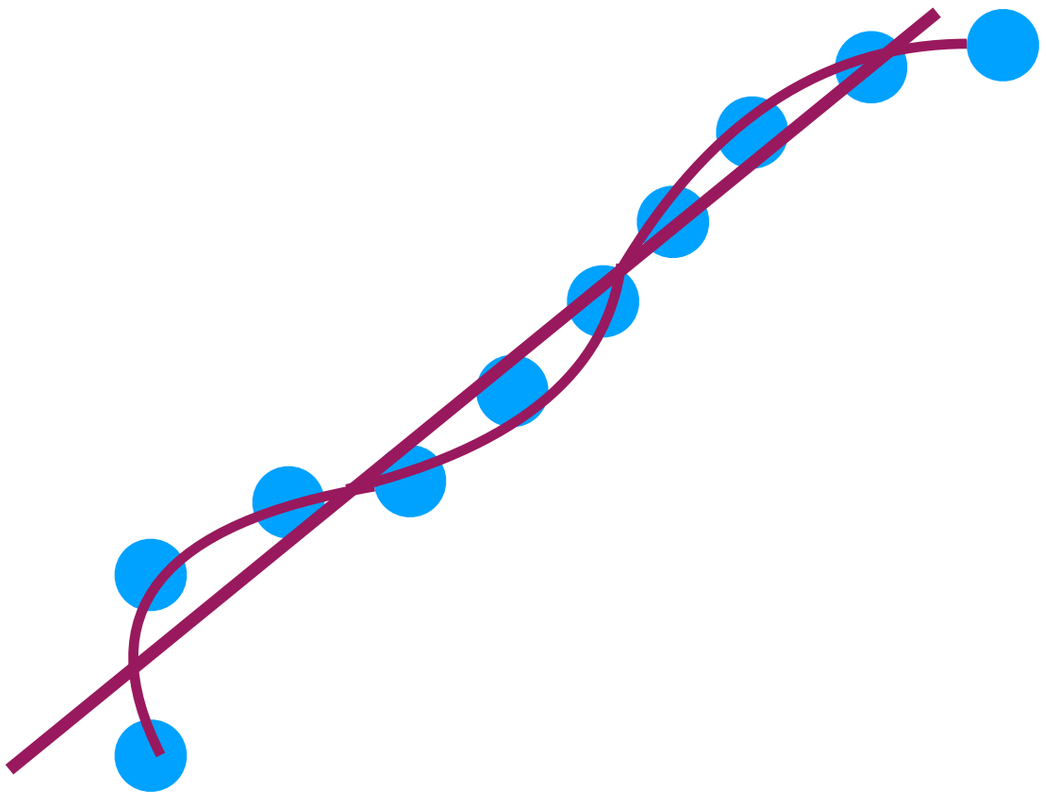
the training & testing performances are close

the training error is low



Need to find the perfect balance!

Occam's Razor



OCCAM'S RAZOR

Anselm BLUMER *
Department of Mathematics and Computer Science, University of Denver, Denver, CO 80208, U.S.A.

Andrzej EHRENFEUCHT **
Department of Computer Science, University of Colorado, Boulder, CO 80302, U.S.A.

David HAUSSLER *
Department of Mathematics and Computer Science, University of Denver, Denver, CO 80208, U.S.A.

Manfred K. WARMUTH ***
Department of Computer and Information Sciences, University of California, Santa Cruz, CA 95064, U.S.A.

Communicated by M.A. Harrison
Received 21 February 1986
Revised 18 July 1986

Occam's Razor

Carl Edward Rasmussen
Department of Mathematical Modelling
Technical University of Denmark
Building 321, DK-2800 Kongens Lyngby, Denmark
carl@imm.dtu.dk http://bayes.imm.dtu.dk

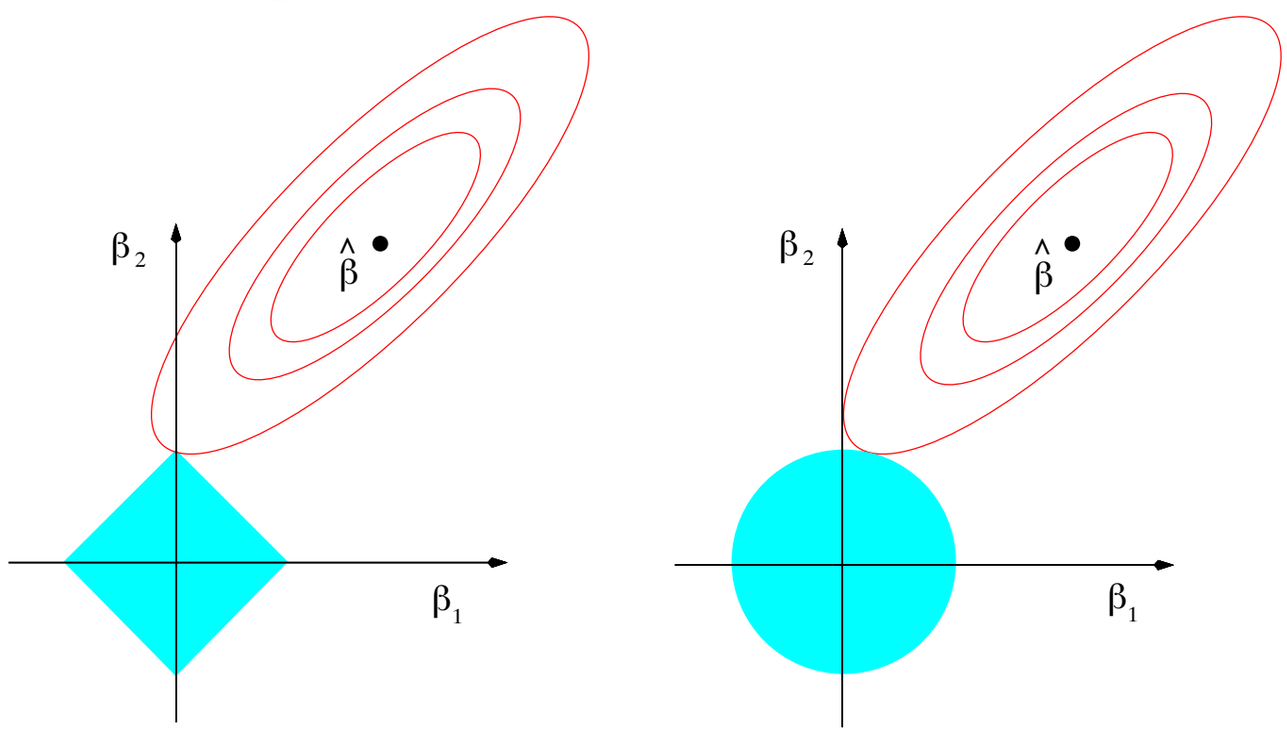
Zoubin Ghahramani
Gatsby Computational Neuroscience Unit
University College London
17 Queen Square, London WC1N 3AR, England
zoubin@gatsby.ucl.ac.uk http://www.gatsby.ucl.ac.uk

Model complexity should be just enough to perform well on training data, but should not be larger than necessary!

Linear Regression with Regularization

- Ridge regression: $\hat{R}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_2^2$
- LASSO: $\hat{R}(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_1$

Regularization terms make weights small



Do linear regression in high dimension and use regularization to control model complexity.
We will learn a strong regularization method in the next lecture!

Machine Learning: I

- Basic concepts
- Nearest neighbor classification
- Linear classification
- Bias-Variance Trade-Off & Regularization
- Take-Home Messages

Take-Home Messages

- Learning = Task + Perf. measure + Data + Algorithm
- Nearest neighbor is lazy learning without training, which has high variance and low bias.
- Linear classification (regression) uses least-square method to optimize MSE, which has low variance and high bias.
- Achieving the balance between variance and bias: regularization.

Thanks for your attention!
Discussions?

Acknowledgement: Many materials in this lecture are taken from
"Elements of Statistical Learning" book
and David Sontag and Joelle Pineau's slides:
https://people.csail.mit.edu/dsontag/courses/ml16/slides/lecture2.pdf
https://www.cs.mcgill.ca/~hvanho2/comp551/slides/02LinearRegression.pdf